Leveraging Large Language Models (LLMs) for Automated Analysis on Indian National Budget Allocation

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Master of Technology
in
Computer Science & Engineering

by

Souparno Ghosh (MT23MCS010)

under the guidance of

Dr. Anil Mokhade

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

Leveraging Large Language Models (LLMs) for Automated Analysis on Indian National Budget Allocation

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of the degree

Master of Technology
in
Computer Science & Engineering

by

Souparno Ghosh (MT23MCS010)

under the guidance of

Dr. Anil Mokhade

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

© Visvesvaraya National Institute of Technology (VNIT) 2025

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology, Nagpur

Declaration

I, **Souparno Ghosh**, hereby declare that this dissertation titled **"Leveraging Large Language Models (LLMs) for Automated Analysis on Indian National Budget Allocation"** is carried out by me in the Department of Computer Science & Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier, wholly or in part, for the award of any degree at this or any other institution.

Souparno Ghosh
Souparno Ghosh
Date: 19/06/25

Certificate

This is to certify that the dissertation titled **"Leveraging Large Language Models (LLMs) for Automated Analysis on Indian National Budget Allocation"** is submitted by **Souparno Ghosh** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science & Engineering**, VNIT Nagpur. The work is comprehensive, complete, and fit for final evaluation.

Dr. Anil Mokhade
Associate Professor
Department of Computer Science & Engineering
VNIT, Nagpur

सह. प्राध्यापक
Associate Professor
संगणक विज्ञान एवं अभियांत्रिकी विभाग
Dept of Computer Science & Engg
वि रा प्रौ सं नागपुर-10 (भारत)
NAGPUR- 0010 (India)

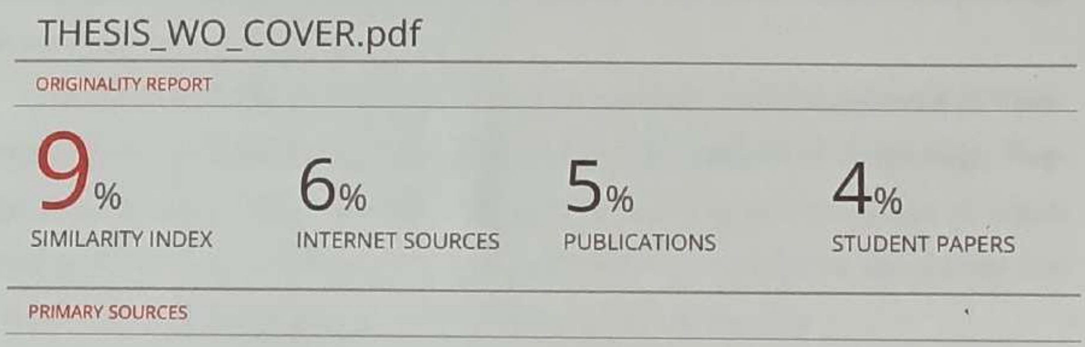
Dr. M.P. Kurhekar
Head, Department of Computer Science & Engineering
VNIT, Nagpur

विभाग प्रमुख
Head
संगणक विज्ञान एवं अभियांत्रिकी विभाग
Dept. of Computer Science & Engg.
वि.रा.प्रौ.सं. नागपुर-१० (भारत)
V.N.I.T., Nagpur-440010. (India)

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology, Nagpur

Non-Plagiarism Certificate

Certified that M. Tech Dissertation titled **"Leveraging Large Language Models (LLMs) for Automated Analysis on National Budget Allocation"** submitted by **Souparno Ghosh**, Enrollment No. **MT23MCS010**, ID No. **30208** has been checked for plagiarism using TURNITIN software and the overlap is found within prescribed limits. A summary of the report is given below.



THESIS_WO_COVER.pdf

ORIGINALITY REPORT

9%	6%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Souparno Ghosh

Souparno Ghosh

Enrollment No: MT23MCS010

Dr. Anil Mokhade

Associate Professor

Department of Computer Science & Engineering

VNIT, Nagpur

सह. प्राध्यापक
Associate Prof., [illegible]
संगणक विज्ञान एवं अभियांत्रिकी विभाग
Dept of Computer Science & Engg
वि रा प्रौ सं नागपुर-10 (भारत)
NAGPUR [illegible]0010 (India)

ACKNOWLEDGEMENT

I would like to take this opportunity to express my heartfelt appreciation to everyone who contributed to the successful completion of my M.Tech dissertation.

First and foremost, I am deeply grateful to my supervisor, **Dr. Anil Mokhade**, for his unwavering guidance, insightful feedback, and continuous encouragement throughout the course of this research. His depth of expertise and thoughtful mentorship have been instrumental in shaping both the direction and quality of this work. It has been a privilege to work under his supervision.

I also extend my sincere thanks to the faculty members of the **Department of Computer Science & Engineering, Visvesvaraya National Institute of Technology, Nagpur**, for providing a strong academic foundation and fostering an environment of critical thinking and research. Their perspectives—shared through coursework, discussions, and reviews—have significantly influenced and enriched this dissertation.

I am especially thankful to my family and friends, whose steadfast support, patience, and motivation have carried me through every stage of this journey. Their belief in my abilities has been a constant source of strength during times of challenge and perseverance.

Thank you.

Souparno Ghosh

Abstract

Leveraging Large Language Models (LLMs) for Automated Analysis on Indian National Budget Allocation Efficient allocation of national budgets across sectors such as healthcare, education, infrastructure, and defense remains a high-stakes, data-heavy challenge. Traditional approaches depend on manual assessments and spreadsheets, often limiting real-time insight and scalability. This study explores the potential of Large Language Models (LLMs) for automating budget analysis through prompt engineering and Retrieval-Augmented Generation (RAG) techniques. By leveraging publicly available budget documents and policy texts, we design a workflow that enables LLMs to extract key trends, flag inconsistencies, and generate sector-wise analytical summaries. While no model fine-tuning was performed, our prompt-based framework demonstrates that LLMs, when properly harnessed, can support more transparent, scalable, and data-driven public financial decision-making.

INDEX

ABSTRACT **i**

INDEX **i**

1 INTRODUCTION **1**

1.1 Overview 1

1.2 Objectives 1

1.3 Thesis Outline 2

2 RELATED WORK **4**

2.1 Introduction 4

2.2 FinMEM: LLM-Based Trading Agent with Layered Memory and Dynamic Character Profiling 4

2.3 FinGPT: Open-Source Financial Large Language Model 5

2.4 FinBERT: Domain-Specific Sentiment Analysis for Financial Texts 6

2.5 Limitations of Existing Works 7

3 PRELIMINARY STUDY **9**

3.1 Transformers 9

3.2 Evolution of the GPT Series and Prompt-Based Adaptation 12

3.2.1 From GPT-2 to GPT-4: Scaling and Capability Emergence 12

3.2.2 Prompt Engineering as a Mechanism for Task Adaptation 12

3.2.3 Hallucination in Large Language Models 13

3.2.4 Conclusion 14

3.3 Retrieval-Augmented Generation (RAG) 14

3.3.1 Model Architecture 15
3.3.2 RAG-Sequence vs. RAG-Token 15
3.3.3 Retriever: Dense Passage Retrieval (DPR) 16
3.3.4 Generator: BART 16
3.3.5 Training Objective 16
3.3.6 Decoding Strategies 17
3.3.7 Relevance to This Work 17
3.4 Word Embedding 17
3.4.1 Word2Vec: CBOW vs. Skip-Gram 18
3.4.2 Training Complexity 18
3.4.3 Advantages of Word2Vec-Based Embedding 19
3.5 Memory Management with Exponential Decay 19
3.5.1 Motivation 20
3.5.2 Mathematical Formulation 20
3.5.3 Archiving Threshold 20
3.5.4 Advantages of Temporal Decay 21
3.5.5 Implementation 21

4 METHODOLOGY **22**
4.1 Proposed Work 22
4.1.1 Memory and Stakeholder Embedding 22
4.1.2 Query Processing and Relevance Matching 23
4.1.3 Language Model Generation 23
4.1.4 Stakeholder Validation 23
4.1.5 Workflow Automation and API Exposure 24
4.2 Algorithm 24

5 Implementation **28**
5.1 Setup 28
5.2 Architecture Overview 30
5.3 Procedure 31

5.4 Performance Tools and Observations 33

6 Results and Observations 35

6.1 Output Snapshots 35

6.1.1 Run 1 36

6.1.2 Run 2 37

6.1.3 Run 3 38

7 Conclusion 39

REFERENCES 42

Chapter 1

INTRODUCTION

1.1 Overview

Large Language Models (LLMs) such as GPT and LLaMA have demonstrated impressive capabilities in natural language understanding, generation, and reasoning across a wide range of general-purpose tasks. Their ability to parse complex text and produce human-like responses has made them increasingly relevant in domains involving large-scale unstructured data, such as governance and public policy.

However, despite their general intelligence, LLMs often struggle with domain-specific accuracy, especially in high-stakes applications like financial analysis or public budgeting. These models tend to hallucinate facts, lack contextual awareness of national frameworks, and fail to reliably retrieve structured information from lengthy policy documents. To address these limitations, this study implements a **Retrieval-Augmented Generation (RAG)** framework on top of a general-purpose LLM. By combining external knowledge retrieval with prompt engineering, we enable the model to focus on relevant sections of national budget documents, thereby improving its ability to analyze allocations, detect inefficiencies, and provide sector-specific insights. This tailored setup enhances the utility of LLMs in financial governance contexts, where precision and interpretability are critical.

1.2 Objectives

This project aims to address the following key questions:

- Can general-purpose Large Language Models (LLMs) extract meaningful insights from national budget documents without domain-specific training?

- How does integrating Retrieval-Augmented Generation (RAG) improve the accuracy and relevance of budget analysis performed by LLMs?

- What are the limitations of prompt-only LLM setups when applied to structured financial data?

- How does the performance of the RAG-enhanced LLM vary across different types of budget sections (e.g., healthcare vs. defense)?

- Can the system reliably identify inefficiencies, inconsistencies, or allocation gaps in real-world budget documents?

  What trade-offs exist between model responsiveness, computational cost, and analytical depth in this setup?

1.3 Thesis Outline

- **Chapter 1** provides an introduction to the project. It highlights the relevance of the chosen topic. The chapter concludes by outlining the key objectives that guide the entire work.

- **Chapter 2** presents the related work and background literature that informed this project. It discusses key advances in Retrieval-Augmented Generation (RAG), embedding models, and domain-specific applications. The chapter establishes a foundation by comparing various retrieval techniques and identifying existing gaps this research seeks to address.

- **Chapter 3** provides a preliminary study that shaped the development of the system. It explores the problem space, analyzes real-world fiscal policy documents, and identifies the limitations of existing LLM responses without retrieval support. This chapter lays the groundwork for the system design decisions made in the implementation phase.

- **Chapter 4** details the methodology adopted for building the Retrieval-Augmented Generation (RAG) system. It outlines the architectural design, selection of models, and data preparation workflows. The chapter provides a structured blueprint that guides the implementation of the proposed system.

- **Chapter 5** describes the implementation of the proposed RAG system in detail. It covers the setup environment, modular architecture, and end-to-end workflow including memory embedding, retrieval, LLM integration, and stakeholder validation. The chapter demonstrates how theoretical design was translated into a functioning prototype.
- **Chapter 6** presents the results and observations from evaluating the RAG system. It analyzes performance metrics, validation outcomes, and retrieval effectiveness across various policy queries. The chapter reflects on how well the system meets its objectives and highlights areas for refinement.

Chapter 2

RELATED WORK

2.1 Introduction

This section reviews recent advancements in applying Large Language Models (LLMs) to financial data analysis. The selected models—FinBERT, FinGPT, and FinMem—demonstrate various approaches to leveraging NLP in finance, each with unique architectures, objectives, and limitations. These works form the comparative foundation for the proposed budget analysis framework.

2.2 FinMEM: LLM-Based Trading Agent with Layered Memory and Dynamic Character Profiling

FinMEM, proposed by Yu et al. (2023) [1], represents a significant evolution in financial large language model (LLM) applications by introducing a cognitively inspired agent architecture for real-time trading. Departing from traditional static LLM pipelines, FinMEM is built as a **modular, autonomous trading agent** with three key subsystems: a profiling module, a layered memory framework, and a decision-making engine. This architecture enables the system to simulate expert human traders by capturing both **time-sensitive market data and long-term contextual memory**.

A core innovation of FinMEM is its **layered memory system**, which emulates human cognitive processing via a combination of working memory (short-term summarization, observation, reflection) and long-term memory stratified into shallow, intermediate, and deep layers. Each memory layer handles different document types—such as daily news, quarterly (10-Q), and annual (10-K) reports—classified by timeliness and retained based on dynamically computed scores of **recency, relevance, and importance**. These scores leverage vector embeddings and attention-guided decay mechanisms, allowing FINMEM to retrieve only the most critical past events during trading decision-making.

Complementing the memory system is the **profiling module**, which establishes a dynamic "character" for each agent instance. It includes both domain-specific knowledge (e.g., industry background, ticker history) and **adaptive risk preferences** (risk-seeking, risk-averse, and self-adjusting). This capability ensures FINMEM can alter its trading behavior based on market volatility or portfolio performance in real time, aligning closely with modern reinforcement learning objectives but framed through a natural language decision logic.

Empirical results across five major stocks (TSLA, NFLX, AMZN, MSFT, COIN) demonstrate FinMEM's **superior cumulative return, Sharpe ratio**, and **drawdown mitigation**, outperforming both traditional DRL agents (DQN, PPO, A2C) and LLM-based baselines like FinGPT and Park et al.'s Generative Agents. Notably, FinMEM achieves these results using **shorter training windows** and
textbfsmaller datasets, emphasizing its design efficiency and contextual robustness.

While FinMEM is primarily focused on autonomous financial trading, its architecture—particularly the **layered memory retrieval** and **adaptive risk-character integration**—offers conceptual value for broader LLM applications in **governance, budget forecasting, and policy alignment**. For this reason, it serves as an important referent in the development of this thesis, which explores prompt-based financial analysis and trend detection in national budget allocations using LLMs.

2.3 FinGPT: Open-Source Financial Large Language Model

FinGPT is an open-source framework introduced by Yang et al. (2023) [2] with the goal of democratizing access to financial large language models (FinLLMs). Developed by the AI4Finance Foundation, FinGPT addresses key challenges in applying LLMs to financial domains, including high temporal sensitivity, low signal-to-noise ratios, and the need for domain-specific adaptation. Unlike proprietary models such as BloombergGPT, which rely on exclusive datasets and expensive training pipelines, FinGPT offers a cost-effective and transparent alternative by adopting a data-centric and modular design.

The FinGPT architecture comprises four layers: a **data source layer** that collects diverse real-time financial data (e.g., news, social media, filings); a **data engineering layer** that processes and cleans data for NLP use; an **LLM layer** that supports fine-tuning through lightweight methods such as LoRA (Low-Rank Adaptation) and RLSP (Reinforcement Learning on Stock Prices); and an **application layer** that demonstrates potential use cases such as robo-advisors, portfolio optimization, sentiment analysis, and fraud detection. The framework emphasizes adaptability, enabling continual updates to remain responsive to rapidly evolving financial markets.

A key contribution of FinGPT is its use of **market-based feedback mechanisms** (e.g., stock price changes) to label and fine-tune financial text data, thereby reducing the reliance on costly human annotations. Additionally, by leveraging modular prompt engineering and supporting low-code deployment, FinGPT significantly lowers the barrier to entry for researchers and developers interested in financial AI.

While FinGPT's real-time processing and affordability mark a significant step toward open financial modeling, its primary focus is trading-oriented and lacks features tailored for public sector governance or policy evaluation. Nonetheless, its retrieval strategies, fine-tuning techniques, and layered memory design offer important inspiration for frameworks aimed at large-scale document analysis and decision-support in economic domains, such as the one proposed in this thesis.

2.4 FinBERT: Domain-Specific Sentiment Analysis for Financial Texts

FinBERT, introduced by Dogu Araci (2019) [3], is one of the earliest efforts to adapt the BERT architecture specifically for financial sentiment analysis. Developed through further pre-training on a financial text corpus, FinBERT addresses the challenge of sentiment classification in finance—a domain characterized by jargon-heavy language, implicit expressions, and scarcity of labeled datasets.

Unlike general-purpose sentiment models, FinBERT is fine-tuned on domain-specific corpora such as the Financial PhraseBank and FiQA datasets. It adopts BERT's base architecture but applies transfer learning strategies like **domain-specific pre-training, slanted**

triangular learning rates, gradual unfreezing, and **discriminative fine-tuning**. These techniques are designed to mitigate catastrophic forgetting during fine-tuning while preserving the deep semantic representations learned from the original BERT pre-training phase.

FinBERT demonstrates **state-of-the-art performance** on financial sentiment classification tasks. On the Financial PhraseBank dataset, it achieved up to **97% accuracy** on fully agreed annotations and significantly outperformed prior baselines such as ULMFit, ELMo, and LSTM-based classifiers. The model also excelled in regression tasks on the FiQA dataset, recording the lowest mean squared error (MSE) and the highest R^2 among contemporary approaches.

A critical contribution of FinBERT is its demonstration that effective results in low-data regimes can be achieved through pre-trained language models, challenging the traditional notion that deep learning models are inherently data-hungry. Additionally, experiments with layer-specific fine-tuning revealed that even fine-tuning only the last few Transformer layers provides competitive performance, reducing training time and compute requirements.

However, FinBERT is narrowly focused on **polarity detection** and does not incorporate memory mechanisms, stakeholder alignment, or retrieval-based reasoning. Its success lies in deep classification, but it lacks the dynamic interaction capabilities required for applications like policy simulation or budget analytics. Nevertheless, it serves as a foundational example of successful **domain adaptation** for LLMs, informing the sentiment-driven components of financial analysis systems like the one proposed in this thesis.

2.5 Limitations of Existing Works

While FinGPT, FinBERT, and FinMEM represent significant progress in applying LLMs to the financial domain, each falls short in critical aspects when evaluated against the unique requirements of **public-sector budget analysis and governance-focused document understanding**.

FinGPT excels in financial text generation and real-time sentiment interpretation but is primarily designed for **trading environments**, not structured policy analysis. Its data-centric pipeline emphasizes fast-changing market indicators, making it ill-suited for interpreting **static, hierarchical documents** such as national budget reports. Moreover, it lacks built-in mechanisms for aligning generated content with stakeholder objectives or policy constraints, which are essential for governance applications.

FinBERT, though highly effective in **sentiment classification**, is limited by its **narrow task orientation**. It is optimized for polarity detection on short financial phrases and lacks both generative capability and multi-document comprehension. FinBERT's static embeddings and absence of a memory or retrieval system make it incapable of handling the layered, context-rich narratives common in budget documents.

FinMEM introduces advanced agent behavior and layered memory for trading simulations but remains tightly coupled to **profit-maximization goals**. Its architecture is tailored to reinforcement learning with risk-based agent profiles, not interpretability or transparency in public finance. Furthermore, while FinMEM's memory structure is innovative, it is deeply bound to **price-reactive signals**, not **explanatory policymaking or multi-stakeholder validation**.

In contrast to these models, this study targets the design of a **lightweight, RAG-based LLM system** that can dynamically retrieve relevant budget segments, align outputs with policy stakeholders, and maintain transparency—all of which are **underrepresented or absent** in the above frameworks.

Chapter 3

PRELIMINARY STUDY

3.1 Transformers

The Transformer architecture, introduced by Vaswani et al. [4], represents a foundational shift in sequence modeling by discarding recurrence in favor of self-attention mechanisms. It adopts an encoder-decoder structure, where both components are composed of stacked layers of attention and feed-forward submodules. Transformers process tokens in parallel rather than sequentially, enabling greater scalability and efficiency in modeling long-range dependencies.

Tokenization and Embeddings

Text input to a Transformer is first tokenized—split into discrete units such as words, subwords, or characters—then mapped to numerical vectors via an embedding matrix. Subword tokenization (e.g., Byte Pair Encoding) is particularly effective for modeling rare or compound words, and is used in models like GPT and BERT. Each token is embedded into a d_{model}-dimensional space. For instance, the original Transformer used 512-dimensional embeddings:

$$\text{Embedding Matrix} \in \mathbb{R}^{d_{\text{model}} \times V}, \quad d_{\text{model}} = 512$$

where V denotes the vocabulary size.

Positional Encoding

Since Transformers lack a recurrent structure, they require a mechanism to capture token order. Positional encodings inject sequence position information into the embeddings

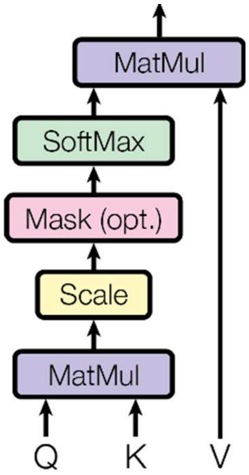


Figure 3.1: Self-Attention

using sinusoidal functions:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

These functions encode position via different frequencies, allowing the model to distinguish both local and global positional dependencies.

Self-Attention Mechanism

Self-attention allows the model to weigh the importance of different tokens in a sequence relative to each other. Each input token is projected into three vectors: a query (Q), a key (K), and a value (V). The attention weights are computed as scaled dot products between queries and keys:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where d_k is the dimensionality of the key vectors. The softmax normalizes these scores into a probability distribution, allowing the model to focus on relevant tokens during representation learning. The values matrix V then aggregates contextual information based on these weights.

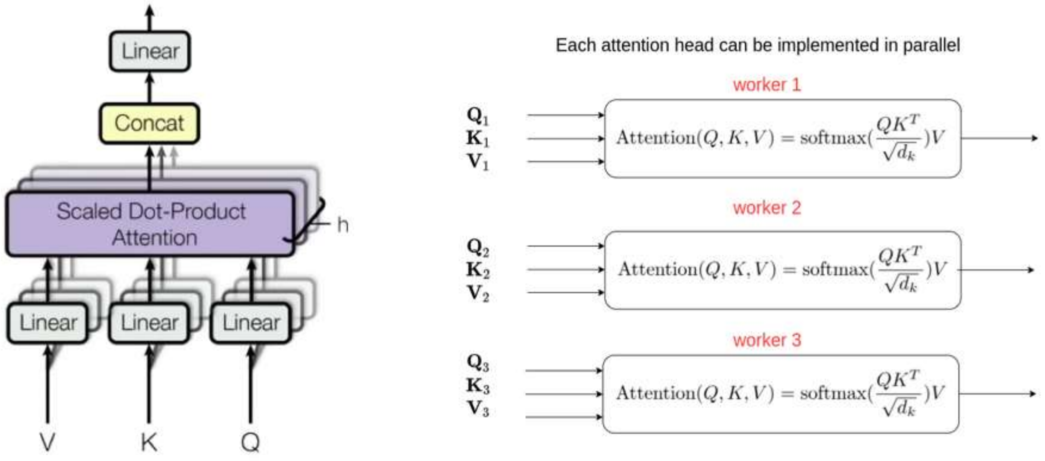

Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Each attention head can be implemented in parallel
worker 1
$\mathbf{Q}_1$
$\mathbf{K}_1$
$\mathbf{V}_1$
$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
worker 2
$\mathbf{Q}_2$
$\mathbf{K}_2$
$\mathbf{V}_2$
$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
worker 3
$\mathbf{Q}_3$
$\mathbf{K}_3$
$\mathbf{V}_3$
$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Figure 3.2: Multi-Head Attention

Multi-Head Attention

To capture diverse aspects of linguistic context, the Transformer employs multiple attention heads. Each head independently applies the attention mechanism using distinct learned projections:

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Here, W_Q^i, W_K^i, W_V^i are projection matrices specific to the i-th head, and W^O is a final output projection matrix. This multi-headed design enables the model to attend to information at different positions and semantic subspaces simultaneously.

Transformers in Practice

Transformer-based models underpin nearly all modern LLMs, including BERT, GPT, T5, and PaLM. Their self-attention-based architecture enables the capture of hierarchical, syntactic, and semantic relationships without recurrence, making them highly effective across tasks such as text classification, translation, summarization, and retrieval-augmented generation. By combining token embeddings, positional encodings, multi-head attention, and deep feed-forward networks, Transformers have set a new standard for scalable and interpretable sequence modeling.

3.2 Evolution of the GPT Series and Prompt-Based Adaptation

3.2.1 From GPT-2 to GPT-4: Scaling and Capability Emergence

The Generative Pretrained Transformer (GPT) series developed by OpenAI has marked successive breakthroughs in natural language generation through architectural scaling and improved pretraining techniques.

GPT-2, released in 2019, consisted of 1.5 billion parameters and was trained using a standard transformer decoder architecture. It employed autoregressive language modeling to predict the next token in a sequence, thereby learning syntactic and semantic patterns from massive web-scale corpora [5]. Despite the absence of fine-tuning, GPT-2 demonstrated coherent zero-shot and few-shot performance across a range of language tasks. However, it suffered from limited factual consistency and poor long-range reasoning.

GPT-3, introduced in 2020, significantly scaled the architecture to 175 billion parameters [6], facilitating emergent capabilities such as in-context learning. In this paradigm, the model learns new tasks from prompts rather than weight updates, enabling zero-shot and few-shot generalization. GPT-3's performance on downstream benchmarks highlighted the power of scale in language models, yet challenges persisted: hallucinations, dataset biases, and limited interpretability continued to restrict applicability in high-stakes domains.

GPT-4, announced in 2023, further advanced the paradigm by integrating multimodal inputs and significantly improving factual accuracy, safety alignment, and multilingual capabilities. Though exact architectural details remain undisclosed, GPT-4 incorporates Reinforcement Learning from Human Feedback (RLHF) [7], chain-of-thought prompting, and better reward modeling to reduce hallucinations and improve instruction-following. Notably, GPT-4 achieved top 10% performance on the U.S. Bar Exam and demonstrated consistent gains across 24 of 26 languages in the MMLU benchmark [8].

3.2.2 Prompt Engineering as a Mechanism for Task Adaptation

Prompt engineering has emerged as a lightweight and interpretable alternative to fine-tuning for task adaptation in large language models. Unlike gradient-based methods, prompt engineering leverages natural language cues to elicit latent capabilities embedded during pretraining [9].

Prompts can be designed in various forms:

- **Direct Task Specification:** Naming the task explicitly (e.g., "Summarize this policy brief").
- **Few-Shot Demonstrations:** Providing examples within the prompt.
- **Chain-of-Thought (CoT) Prompts:** Structuring reasoning in multiple steps to enhance factual consistency [10].
- **Metaprompts:** Prompts that instruct the model to generate its own task-specific instructions.
- **Expert Emulation:** Framing the model as a specific persona (e.g., "You are a fiscal analyst").

These techniques allow LLMs to generalize across domains such as legal reasoning, policy synthesis, and financial forecasting without incurring the computational overhead of fine-tuning. For instance, a well-phrased prompt like *"Translate French to English:"* can yield high-quality results using only the model's pretraining knowledge, assuming adequate exposure to translation pairs during training.

3.2.3 Hallucination in Large Language Models

Despite their fluency and versatility, large language models are prone to **hallucinations**—confident but factually incorrect or unverifiable outputs. These arise primarily from the model's reliance on statistical co-occurrence rather than grounded reasoning or factual databases. Hallucinations are particularly dangerous in domains requiring factual consistency, such as fiscal governance or legal documentation.

Ji et al. (2023) [11] categorize hallucinations into intrinsic and extrinsic types:

- **Intrinsic Factors:** Training data quality, model architecture, and parameter scale.
- **Extrinsic Factors:** Input ambiguity, prompt design, and task complexity.

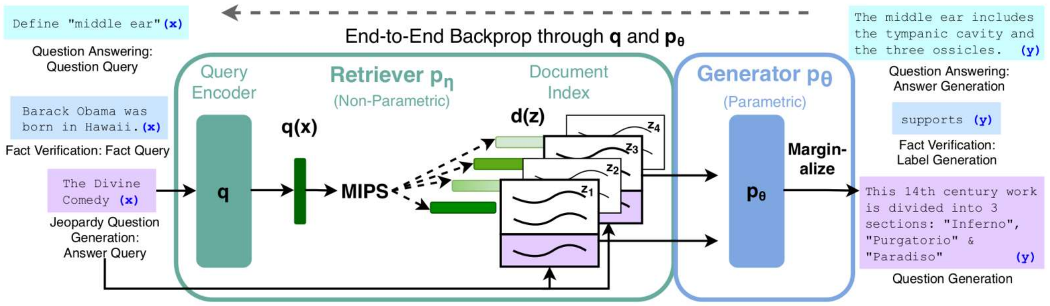
Although techniques like Retrieval-Augmented Generation (RAG), CoT prompting, and RLHF help mitigate hallucinations, the problem remains unsolved. RAG enhances grounding by retrieving relevant documents [12], while CoT prompts improve logical coherence by breaking down complex reasoning tasks into interpretable steps.

3.2.4 Conclusion

The evolution from GPT-2 to GPT-4 showcases the scaling laws of deep learning, where increased parameters and data volumes lead to emergent behaviors such as few-shot learning, chain-of-thought reasoning, and multimodal alignment. Prompt engineering has emerged as a scalable, flexible method for leveraging pretrained knowledge without retraining, enabling effective domain adaptation in contexts like law, policy, and finance.

Nevertheless, key limitations persist: hallucination, lack of persistent memory, and finite context windows constrain reliability in decision-critical applications. Ongoing research in grounded generation, memory-augmented models, and self-verification mechanisms is vital for ensuring that LLMs can be safely and effectively integrated into real-world governance and analytical pipelines.

3.3 Retrieval-Augmented Generation (RAG)



Define "middle ear" (x)
Question Answering: Question Query
Barack Obama was born in Hawaii. (x)
Fact Verification: Fact Query
The Divine Comedy (x)
Jeopardy Question Generation: Answer Query
End-to-End Backprop through q and p_θ
Query Encoder
Retriever p_η
(Non-Parametric)
Document Index
q
q(x)
MIPS
d(z)
z_4
z_3
z_2
z_1
Generator p_θ
(Parametric)
p_θ
Margin-alize
The middle ear includes the tympanic cavity and the three ossicles. (y)
Question Answering: Answer Generation
supports (y)
Fact Verification: Label Generation
This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso" (y)
Question Generation

Figure 3.3: Overview of RAG

Retrieval-Augmented Generation (RAG) is a hybrid neural architecture that integrates large language models (LLMs) with a non-parametric memory retrieval mechanism. Originally proposed by Lewis et al. (2020) [12], RAG is designed to address the limitations of parametric-only models by augmenting generation with relevant external knowledge. This is especially beneficial for knowledge-intensive tasks such as policy analysis, question answering, and financial document summarization, where factual precision and contextual relevance are critical.

RAG leverages two core components: a **retriever** that selects relevant documents from an external corpus, and a **generator** that produces output conditioned on both the input and retrieved context. Together, they form a latent variable model that jointly reasons over learned parameters and retrieved knowledge.

3.3.1 Model Architecture

The RAG framework combines two probabilistic submodels:

- **Retriever** $p_\eta(z \mid x)$: A dense retrieval mechanism that computes the probability of selecting a document z given an input query x. It encodes both documents and queries into a shared embedding space, facilitating semantic matching.
- **Generator** $p_\theta(y_i \mid x, z, y_{1:i-1})$: A sequence-to-sequence language model (e.g., BART) that generates each token y_i conditioned on the input x, the retrieved document z, and the previously generated tokens $y_{1:i-1}$.

These modules interact through a latent variable formulation where predictions are marginalized over the top-k retrieved documents. This enables the model to incorporate multiple relevant knowledge sources in a probabilistic manner.

3.3.2 RAG-Sequence vs. RAG-Token

RAG supports two inference strategies:

- **RAG-Sequence:** Assumes a single retrieved document z is responsible for the entire output sequence y. The final output distribution marginalizes over the top-k retrieved documents:

  $$p(y \mid x) \approx \sum_{z \in \text{top-}k} p_\eta(z \mid x) \cdot p_\theta(y \mid x, z)$$

- **RAG-Token:** Allows each output token y_i to be generated using a potentially different document z_i, enabling fine-grained context selection:

$$p(y \mid x) \approx \prod_{i} \sum_{z \in \text{top-}k} p_\eta(z \mid x) \cdot p_\theta(y_i \mid x, z, y_{1:i-1})$$

While RAG-Token offers more flexibility and improved performance on multi-hop reasoning, it incurs higher computational cost during inference.

3.3.3 Retriever: Dense Passage Retrieval (DPR)

The retriever component employs **Dense Passage Retrieval (DPR)**, a bi-encoder architecture. Documents and queries are encoded separately using BERT-based encoders into dense vectors $d(z)$ and $q(x)$ respectively. Relevance is computed using the inner product:

$$p_\eta(z \mid x) \propto \exp(d(z)^\top q(x))$$

Document retrieval is performed using Approximate Maximum Inner Product Search (MIPS) implemented via FAISS, allowing efficient top-k selection from millions of documents.

3.3.4 Generator: BART

The generator module is based on the BART-large model, a pre-trained encoder-decoder transformer. At generation time, the retrieved document z is concatenated with the input x and passed to BART, which outputs the target sequence y autoregressively.

3.3.5 Training Objective

RAG is trained using the marginal log-likelihood over the retrieved documents:

$$\mathcal{L} = -\log \sum_{z \in \text{top-}k} p_\eta(z \mid x) \cdot p_\theta(y \mid x, z)$$

During training, the document encoder $d(z)$ is typically kept fixed, while the query encoder $q(x)$ and the generator parameters θ are fine-tuned. This enables end-to-end learning without requiring explicit supervision on which document is correct.

3.3.6 Decoding Strategies

At inference time, RAG supports two decoding approaches:

- **RAG-Token Decoding:** Beam search is performed with token-level marginalization across documents, offering token-specific conditioning.
- **RAG-Sequence Decoding:** Beam search is conducted independently for each retrieved document. The resulting candidate sequences are then rescored based on their marginal likelihood.

Two practical modes of execution exist:

- **Thorough Decoding:** Exhaustive beam re-ranking across documents.
- **Fast Decoding:** Uses heuristics to prune low-probability beams early.

3.3.7 Relevance to This Work

In this thesis, RAG serves as the foundational framework for integrating memory into the language generation pipeline. The design enables semantic retrieval from fiscal documents, stakeholder statements, and regulatory corpora, using document embeddings generated from domain-specific corpora. Subsequent generations are enriched by retrieved context, ensuring factual alignment and minimizing hallucination—a critical requirement in public policy and finance applications.

3.4 Word Embedding

Word embeddings are dense vector representations that encode semantic relationships between words based on their co-occurrence patterns in natural language. Unlike traditional one-hot encoding or bag-of-words (BoW) models—which represent words as sparse and independent vectors—embeddings place semantically similar words close to each other in a continuous high-dimensional vector space. This property enables models to generalize meaning across similar terms and improves performance on downstream natural language processing (NLP) tasks.

In memory-augmented architectures such as Retrieval-Augmented Generation (RAG), word embeddings serve multiple roles:

- Representing stored documents (e.g., fiscal policies, budget reports) and stakeholder agendas.
- Embedding incoming queries and generated responses into the same vector space.
- Enabling semantic retrieval via cosine similarity, allowing for “soft matching” even when the query and memory use different but related terms.

3.4.1 Word2Vec: CBOW vs. Skip-Gram

Word2Vec, proposed by Mikolov et al. [13], introduced two neural architectures for learning word embeddings:

Continuous Bag of Words (CBOW): CBOW predicts a target word given its surrounding context. It is computationally efficient and well-suited for large corpora but often underperforms when dealing with rare or domain-specific terms due to its averaging of contextual signals.

Skip-Gram: In contrast, the Skip-Gram model predicts the surrounding context given a target word. This architecture performs better on sparse or specialized datasets because it can capture more meaningful representations for low-frequency terms. This makes it highly suitable for domain-specific corpora, such as fiscal policy documents or economic briefings, where uncommon but critical vocabulary occurs.

3.4.2 Training Complexity

The computational complexity of training Word2Vec varies between the two architectures:

- CBOW:

$$Q = N \times D + D \times \log_2(V)$$

- Skip-Gram:

$$Q = C \times (D + D \times \log_2(V))$$

Where:

- N: size of the context window (CBOW)
- C: maximum skip distance (Skip-Gram)
- D: dimensionality of the word vectors
- V: size of the vocabulary

In this project, the Skip-Gram model is used due to its superior performance on rare and context-specific terminology often found in policy-oriented datasets. It enables richer embedding representations for low-frequency but semantically important tokens, making it more robust under data sparsity.

3.4.3 Advantages of Word2Vec-Based Embedding

The Word2Vec framework offers multiple advantages for embedding-centric systems:

- **Efficiency:** The Skip-Gram model can be trained on millions of tokens within hours using modest hardware.
- **Semantic Richness:** Embeddings capture analogical relationships such as:

$$\text{“king”} - \text{“man”} + \text{“woman”} \approx \text{“queen”}$$

- **Empirical Performance:** Word2Vec embeddings have outperformed earlier neural language models such as NNLM and RNNLM, as well as CBOW, in various semantic tasks—often achieving over 55% accuracy.
- **Scalability:** Embedding performance improves with increased dimensionality and data size, scaling effectively up to 1000 dimensions and corpora with 6 billion+ tokens.

3.5 Memory Management with Exponential Decay

To maintain the system's long-term accuracy and efficiency, a memory decay logic [14] is employed to filter outdated or low-relevance information. This prevents stale data from dominating recent, query-relevant embeddings and ensures that generated responses remain temporally contextual and semantically fresh.

3.5.1 Motivation

In a Retrieval-Augmented Generation (RAG) framework, memory embeddings accumulate over time. Without pruning, older documents—regardless of relevance—can still rank highly due to lexical similarity, particularly for generic or ambiguous queries. To address this, the system applies a time-weighted decay to cosine similarity scores, discounting memory vectors that have not been recently activated.

3.5.2 Mathematical Formulation

Given a memory embedding $\vec{m}_i$ and a current query embedding $\vec{q}$, the raw cosine similarity is computed as:

$$\text{cos\_sim}(\vec{q}, \vec{m}_i) = \frac{\vec{q} \cdot \vec{m}_i}{\|\vec{q}\| \cdot \|\vec{m}_i\|}$$

To incorporate temporal relevance, this similarity is adjusted using an exponential decay function:

$$\text{AdjustedScore}_i = \text{cos\_sim}(\vec{q}, \vec{m}_i) \cdot e^{-\lambda t_i}$$

Where:

- t_i is the time elapsed since $\vec{m}_i$ was last activated (e.g., in days or query intervals)
- λ is the decay constant, typically in the range $0.1 \leq \lambda \leq 0.5$

This formulation ensures that memory entries unused for extended periods will have lower effective scores, reducing their influence during retrieval.

3.5.3 Archiving Threshold

The adjusted score AdjustedScore_i is compared to a predefined minimum threshold Th_{decay}. If the score falls below this threshold:

$$\text{AdjustedScore}_i < Th_{\text{decay}} \Rightarrow \text{archive or delete } \vec{m}_i$$

Such vectors are archived (rather than permanently deleted) for traceability and audit purposes but are excluded from future similarity computations.

3.5.4 Advantages of Temporal Decay

Incorporating exponential decay into memory scoring offers several operational and strategic benefits. First, it significantly reduces memory bloat by demoting outdated or contextually stale embeddings, ensuring that the active memory index remains compact and relevant. This not only optimizes storage but also accelerates retrieval operations. Second, it improves semantic precision by emphasizing documents that are temporally closer to the current query, thus surfacing fresher and more contextually aligned content. Third, by periodically attenuating the influence of old embeddings, the system maintains a high degree of responsiveness, making it more suitable for real-time and streaming applications such as fiscal monitoring or policy impact assessments.

3.5.5 Implementation

The memory decay mechanism can be implemented using a lightweight backend infrastructure. Relevance scores and associated timestamps are stored in either JSON or SQLite formats, allowing for fast lookup and update operations. A scheduled cron job executes at regular intervals (e.g., every 24 hours) to recompute adjusted scores for all active memory entries based on the elapsed time since their last activation. To maintain performance and reduce overhead, the embedding vectors are updated in-place rather than reindexed, minimizing disk I/O during decay propagation. This design ensures that the system remains efficient without sacrificing retrieval accuracy over time.

Chapter 4

METHODOLOGY

4.1 Proposed Work

This work presents a Retrieval-Augmented Generation (RAG) framework designed to assist policymakers in analyzing national budget data and generating stakeholder-aware responses using large language models (LLMs). The system integrates document embeddings, stakeholder alignment logic, and dynamic context construction to ensure semantically relevant and politically aligned outputs.

4.1.1 Memory and Stakeholder Embedding

The system begins by embedding two primary information sources into a shared vector space:

- **Memory Documents**: These include fiscal policy papers, budget reports, GDP datasets, and global economic indicators.
- **Stakeholder Agendas**: These consist of position papers and policy directives from bodies such as the Finance Ministry, labor unions, SMEs, and environmental groups.

Text from these sources is preprocessed and vectorized using both Word2Vec (specifically the Skip-Gram architecture) and Sentence Transformers (e.g., `all-mpnet-base-v2`). The resulting dense embeddings are stored for efficient semantic similarity matching during query processing.

4.1.2 Query Processing and Relevance Matching

When a user submits a natural language query, it is embedded into the same vector space as the memory corpus. To identify relevant memory entries, cosine similarity is computed between the query vector $\vec{q}$ and each memory vector $\vec{m}_i$:

$$\text{cos\_sim}(\vec{q}, \vec{m}_i) = \frac{\vec{q} \cdot \vec{m}_i}{||\vec{q}|| \cdot ||\vec{m}_i||}$$

To prioritize more recent or contextually active information, the similarity score is adjusted using an exponential decay function:

$$\text{AdjustedScore}(i) = \cos(\theta_i) \cdot e^{-\lambda t_i}$$

Where:

- θ_i is the angle between $\vec{q}$ and $\vec{m}_i$
- t_i is the time elapsed since $\vec{m}_i$ was last activated
- λ is the decay constant (hyperparameter, typically $0.1 \leq \lambda \leq 0.5$)

Only those memory vectors whose decayed score exceeds a defined threshold are retained and concatenated to form the dynamic context passed to the language model.

4.1.3 Language Model Generation

The selected memory segments, along with the user query, are passed as input to a large language model (e.g., GPT-3.5, GPT-4) or a locally hosted, quantized model such as Phi-2. The language model is instructed to generate responses *only* based on the given context, thereby reducing hallucinations and enforcing factual accuracy.

For local inference, the system supports models served via the HuggingFace Transformers pipeline, enabling cost-efficient and offline deployments.

4.1.4 Stakeholder Validation

After the LLM generates a candidate response, it is embedded as a vector $\vec{a}$. The response vector is then compared to each stakeholder agenda vector $\vec{s}_j$ using cosine similarity:

$$\text{cos\_sim}(\vec{a}, \vec{s}_j) \geq Th_s \Rightarrow \text{Response is stakeholder-aligned}$$

Only responses that meet or exceed the alignment threshold Th_s for at least one stakeholder are presented to the user. This filtering mechanism ensures that the outputs are not only semantically correct but also politically or organizationally appropriate.

4.1.5 Workflow Automation and API Exposure

The system is implemented in a modular and extensible fashion, consisting of the following components:

- **Memory Module**: Handles embedding, decay computation, and semantic similarity search.
- **LLM Module**: Constructs prompts and manages interaction with the LLM API or local model.
- **Stakeholder Module**: Validates output alignment with policy agendas using cosine similarity.

Supporting tools such as `pandas`, `scipy`, and `nltk` are employed for preprocessing, document parsing, and tokenization.

4.2 Algorithm

This section outlines the core algorithms that power the proposed Retrieval-Augmented Generation (RAG) framework for stakeholder-aware policy response generation. The system is structured around three interlinked processes:

- **Algorithm 1** describes the embedding pipeline that converts memory documents and stakeholder agendas into dense vector representations for downstream semantic retrieval.
- **Algorithm 2** details the query processing and response generation flow, integrating retrieval, LLM invocation, and stakeholder validation.

- **Algorithm 3** introduces a temporal decay-based memory scoring mechanism to prioritize fresher and more contextually relevant memory vectors during retrieval.

Together, these algorithms ensure that the system produces context-grounded, stakeholder-aligned outputs while maintaining retrieval efficiency and temporal relevance.

Algorithm 1 Embedding Creation for Memory and Stakeholder Agendas

```
Require: Document set $D$ (e.g., national budget reports, economic surveys)
Require: Stakeholder agenda set $S$ (e.g., policy mandates, position papers)
Require: Pretrained embedding model $M$ (e.g., all-mpnet-base-v2, Word2Vec)
Ensure: Memory embeddings $E_{\text{memory}}$
Ensure: Stakeholder embeddings $E_{\text{agendas}}$
Initialize text lists $T_{\text{memory}} \leftarrow [\,]$, $T_{\text{agendas}} \leftarrow [\,]$
for each document $d \in D$ do
    Convert PDF to text if needed, store as $t_d$
    Append $t_d \rightarrow T_{\text{memory}}$
end for
for each agenda $s \in S$ do
    Read text content $t_s$
    Append $t_s \rightarrow T_{\text{agendas}}$
end for
for each $t \in T_{\text{memory}}$ do
    $e_t \leftarrow$ Encode$(M, t)$
    Append $e_t \rightarrow E_{\text{memory}}$
end for
for each $a \in T_{\text{agendas}}$ do
    $e_a \leftarrow$ Encode$(M, a)$
    Append $e_a \rightarrow E_{\text{agendas}}$
end for
Persist $E_{\text{memory}}, E_{\text{agendas}}$ to disk as .npy or .json
```

Algorithm 2 RAG-Based Query Answering and Stakeholder Alignment

```
Require: User query $q$
Require: Memory embeddings $E_{\text{memory}}$ and texts $T_{\text{memory}}$
Require: Stakeholder embeddings $E_{\text{agendas}}$
Require: Language model $M$
Ensure: Final stakeholder-approved response $r$
Encode query: $e_q \leftarrow \text{Encode}(M, q)$
Compute cosine similarity with memory:
for each $e_m \in E_{\text{memory}}$ do
    $\text{sim}(e_q, e_m) \leftarrow \cos(e_q, e_m)$
end for
Select top-$k$ memory items with highest similarity
Construct context: $C \leftarrow \text{concat}(T_{\text{top-k}})$
Generate initial response: $r_{\text{initial}} \leftarrow \text{Generate}(M, C, q)$
Encode response: $e_r \leftarrow \text{Encode}(M, r_{\text{initial}})$
Check stakeholder alignment:
for each $e_a \in E_{\text{agendas}}$ do
    Compute $\text{sim}(e_r, e_a)$
end for
if $\forall e_a,\ \text{sim}(e_r, e_a) \geq \theta$ then
    Accept response
    $r \leftarrow r_{\text{initial}}$
else
    Optionally retry with more context or fallback model
    $r \leftarrow r_{\text{final}}$
end if
return $r$
```

Algorithm 3 Memory Scoring with Exponential Decay

```
Require: Query embedding $\vec{e}_q$
Require: Memory embeddings $\{\vec{e}_i\}_{i=1}^{N}$
Require: Last activation timestamps $\{t_i\}_{i=1}^{N}$
Require: Decay rate $\lambda$
Require: Threshold $Th_{\text{decay}}$
Ensure: Active memory index set $\mathcal{A} \subseteq \{1, 2, ..., N\}$
Initialize active set: $\mathcal{A} \leftarrow \emptyset$
for each $i = 1$ to $N$ do
    Compute cosine similarity:
        $$\text{sim}_i \leftarrow \frac{\vec{e}_q \cdot \vec{e}_i}{\|\vec{e}_q\| \cdot \|\vec{e}_i\|}$$
    Apply exponential decay:
        $$\text{score}_i \leftarrow \text{sim}_i \cdot \exp(-\lambda \cdot t_i)$$
    if $\text{score}_i \geq Th_{\text{decay}}$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
    end if
end for
return $\mathcal{A}$
```

Chapter 5

IMPLEMENTATION

5.1 Setup

The Retrieval-Augmented Generation (RAG) system is designed to enhance the capabilities of natural language processing by integrating a retrieval mechanism with generative models. This system has been implemented and tested under the following environment and configuration:

- **Operating System:** Ubuntu 22.04 LTS (Local) The development environment for this system was set up on a local machine running Ubuntu 22.04 LTS, a stable and well-supported operating system that is commonly used in data science and machine learning workflows due to its robustness and compatibility with many popular tools and libraries.
- **Programming Language:** Python 3.10 Python 3.10 was chosen as the primary programming language due to its strong support for machine learning and natural language processing (NLP) libraries. This version of Python provides improved performance and syntax features which are ideal for developing complex models like the RAG system.
- **Frameworks and Libraries:** The system leverages several powerful frameworks and libraries to handle various aspects of the implementation:
    - `Transformers` – for the integration of state-of-the-art transformer-based models such as GPT and BERT, essential for generative tasks.

  - `SentenceTransformers` – to generate sentence-level embeddings which are crucial for the retrieval mechanism.
  - `Gensim` – used for additional word embedding models such as Word2Vec, which assist in creating dense vector representations of textual data.
  - `scikit-learn`, `scipy`, `numpy`, and `pandas` – these libraries handle various data manipulation, machine learning, and statistical tasks like cosine similarity scoring, which is key in the retrieval process.
  - `openai` – provides the API interface to OpenAI's GPT models (such as GPT-3.5-Turbo), which are used for the text generation step in the RAG pipeline.
  - `tqdm` – used for displaying progress bars during long-running tasks like model inference and document retrieval.
  - `dotenv` – used for managing environment variables, particularly for securely storing API keys and other sensitive information.
  - `matplotlib` – used for visualizing cosine similarity distributions and fine-tuning similarity thresholds during the evaluation phase.

- **Hardware Configuration:** The system's hardware was configured as follows:
  - **Local Development:** The local machine is equipped with an AMD Ryzen 7 7840HS CPU, 32 GB of RAM, and no dedicated GPU. This configuration is suitable for testing the system on a smaller scale or for initial prototype development. However, for larger datasets or more intensive operations, scaling up the hardware or using cloud-based solutions would be necessary.
  - **Embedding Models:** Sentence Transformers, such as `all-mpnet-base-v2`, were used for generating dense vector embeddings for both memory documents and user queries. Additionally, a local fallback embedding model, `microsoft/phi-2`, is used for offline inference when a GPU is unavailable.

5.2 Architecture Overview

The architecture of the system is built around the concept of Retrieval-Augmented Generation (RAG), where the system first retrieves relevant information from a memory store and then augments that information with answers generated by an LLM (Language Model). The key capabilities of the RAG pipeline include:

1. Embedding and storing both memory documents (such as reports, datasets, and stakeholder agendas) and their corresponding vector representations (embeddings). These embeddings are dense, high-dimensional vectors that represent the semantic content of the documents.
2. Retrieving the top-k most relevant documents from the memory store based on cosine similarity between the embedded query and the stored document embeddings. This allows the system to find and return the most pertinent information to answer the user query.
3. Constructing context-aware prompts by combining the user's query with the relevant documents retrieved from memory. This ensures that the generated answer is grounded in factual data and aligns with the query context.
4. Generating answers using a pre-trained LLM (e.g., GPT-3.5-Turbo) based on the context-aware prompt and validating the generated answers against the stored stakeholder vectors to ensure alignment with stakeholder priorities and policies.
5. Filtering the responses based on alignment scores, ensuring that the generated answer passes specific thresholds for relevance and accuracy according to stakeholder documents (e.g., from the Finance Ministry, Labor Unions, etc.).

The system is modular, with various components split into separate scripts to handle distinct tasks:

1. `llm-wrapper.py` – this script is responsible for formatting the query and memory documents into a suitable prompt, and managing interactions with the LLM to generate answers.

2. `memory-store.py` – this script handles the storage and indexing of the memory documents, as well as the vectorization of these documents using embedding models such as Sentence Transformers and Word2Vec.
3. `retriever.py` – performs the core retrieval task, comparing the user's query to the stored document embeddings and selecting the most relevant memory chunks based on cosine similarity scores.
4. `evaluate-rag.py` – orchestrates the end-to-end process, handling query processing, generating responses, logging the results, and performing stakeholder validation checks on the generated answers.

5.3 Procedure

The implementation follows a step-by-step procedure for handling queries, embedding documents, performing retrieval, generating answers, and validating results. The procedures outlined below describe the processes in detail:

Procedure 1: Memory Embedding and Storage

In this procedure, the system processes and embeds the memory documents into vector representations to allow for efficient retrieval:

- Word2Vec (Skip-Gram) and Sentence Transformers (`all-mpnet-base-v2`) are employed to generate dense, high-quality embeddings that capture the semantic meaning of the documents. These embeddings serve as the foundation for the retrieval process.
- The memory corpus includes a variety of documents such as NITI Aayog reports, IMF policy briefs, and World Bank GDP datasets. These documents provide a rich dataset for answering fiscal policy-related queries.
- The embeddings are serialized and stored in JSON or NumPy formats, making them easy to retrieve and manage efficiently during the query process. This format ensures quick access and retrieval speeds.

Procedure 2: Query Similarity Scoring

This procedure involves embedding the user's query and comparing it to the pre-embedded memory documents:

- The user's natural language query is processed by an embedding model and transformed into a dense vector representation.
- The query vector is compared to all stored memory vectors using cosine similarity, which measures the angle between vectors and indicates how similar they are.
- A cosine similarity threshold is fine-tuned to balance precision and recall, ensuring that only the most relevant documents are retrieved.
- The top-k most relevant memory chunks are selected for each query based on the similarity scores. This subset of documents is then used to generate the final answer.

Procedure 3: Answer Generation with LLM

Once relevant documents are retrieved, they are passed to the LLM for answer generation:

- The selected memory chunks, along with the original user query, are combined into a context-aware prompt. This ensures that the LLM receives enough information to generate an accurate and relevant answer.
- The prompt is sent to OpenAI's GPT-3.5-Turbo API for text generation. The model processes the input and returns a response, which is the system's answer to the user's query.
- In cases where the GPT-3.5-Turbo API is unavailable, a local fallback option is used, leveraging the `microsoft/phi-2` model for offline inference on the local machine. This ensures system functionality even without an internet connection.

Procedure 4: Stakeholder Validation

After generating a response, the system ensures that the answer aligns with relevant stakeholder perspectives:

- Documents from key stakeholders, such as the Finance Ministry, Labor Unions, and SMEs, are embedded using the same embedding models to ensure consistent representation of their viewpoints.
- The generated answer is encoded into a vector and compared to the stakeholder embeddings. This step evaluates whether the answer is in line with stakeholder values and policies.
- The answer is only accepted if the cosine similarity score exceeds a pre-set threshold that ensures the response aligns with stakeholder concerns. If the threshold is not met, the answer is rejected, and the system may attempt to generate a new response.

5.4 Performance Tools and Observations

During testing and evaluation, several tools and observations were made to ensure the system's performance:

- The inference speed on the `phi-2` model, running on the local CPU, was approximately 4 seconds per 256-token response. This performance is acceptable for small-scale testing, though improvements would be necessary for larger datasets or more complex queries.
- Cosine similarity distributions were visualized using `matplotlib` to better understand the retrieval performance and fine-tune similarity thresholds for optimal document retrieval.
- All experiment outputs were logged in `rag`$_{r}esult.jsonfiles, whichcaptureddetailedinformat$

    - Retrieved document chunks
    - The final LLM-generated response
    - Cosine similarity scores
    - Heuristic hallucination proxy scores (to measure the factuality of the response)

This data is used for further refinement of the system, ensuring that it provides the most accurate and aligned responses possible.

Chapter 6

RESULTS AND OBSERVATIONS

6.1 Output Snapshots

The following output snapshots represent the JSON responses generated by the Retrieval-Augmented Generation (RAG) pipeline. Each listing corresponds to a distinct query run, labeled sequentially (Run 1, Run 2, Run 3), and includes detailed output fields.

Each JSON block is composed of the following elements:

- **query:** The natural language question submitted to the RAG system.
- **retrieved_chunks:** A list of top-ranked context chunks retrieved from the document corpus, based on semantic similarity to the query.
- **model_answer:** The generated answer from the language model, conditioned on the query and the retrieved documents.
- **relevance_scores:** Numerical similarity scores (cosine distance) between the query and each retrieved chunk, indicating contextual relevance.
- **hallucination_score:** A scalar value (0 to 1) estimating the factual reliability of the model's response, where values closer to 1 indicate lower hallucination risk.
- **pass_fail:** A binary indicator representing whether the generated answer passed heuristic or manual evaluation criteria for factual accuracy and relevance.

Each output has been listed with a caption and figure label for reference. The content is formatted in `lstlisting` blocks for improved readability and reproduction of exact outputs.

6.1.1 Run 1

```
{
  "query": "What are the three core pillars of India\s
   21st-century fiscal architecture, and how do they contribute
   to fiscal sustainability?",
  "retrieved_chunks": [
    "Indias Fiscal Architecture",
    "Enhancing Fiscal Transparency and Reporting in India",
    "GDP and Beyond: Measuring Economic Progress and
   Sustainability",
    "Global Economic Prospects January 2021: Regional Overview
   SAR",
    "GDP as a Measure of Economic Well-being"
  ],
  "model_answer": "The three core pillars of India's
   21st-century fiscal architecture are enhancing fiscal
   transparency and reporting, measuring economic progress and
   sustainability beyond GDP, and utilizing GDP as a measure of
   economic well-being. These pillars contribute to fiscal
   sustainability by promoting accountability, ensuring
   comprehensive assessment of economic development, and guiding
   policy decisions based on a holistic understanding of
   economic well-being.",
  "relevance_scores": [0.8453, 0.7176, 0.4317, 0.4183, 0.3965],
  "hallucination_score": 0.8063,
  "pass_fail": "Pass"
}
```

Listing 6.1: Output JSON of RAG 1 from RAG pipeline

6.1.2 Run 2

```
{
  "query": "What is the link between education and economic
   productivity in India?",
  "retrieved_chunks": [
    "Education and Economic Growth",
    "Determinants of Economic Growth A Bayesian Panel Data
   Approach",
    "Global Economic Prospects January 2021: Regional Overview
   SAR",
    "South Asian Economic Focus",
    "How Economic Development Influences The Environment"
  ],
  "model_answer": "Education plays a crucial role in enhancing
   economic productivity in India. Studies have shown that a
   well-educated workforce is more productive, innovative, and
   adaptable to changes in the economy. Higher levels of
   education lead to higher skills and knowledge levels, which
   in turn contribute to increased efficiency and effectiveness
   in the workforce. Additionally, education can also lead to
   higher levels of entrepreneurship and technological
   advancement, further boosting economic productivity in the
   country.",
  "relevance_scores": [0.6403, 0.4424, 0.4410, 0.3966, 0.3924],
  "hallucination_score": 0.6584,
  "pass_fail": "Pass"
}
```

Listing 6.2: Output JSON of RAG 2 from RAG pipeline

6.1.3 Run 3

```
{
  "query": "Which government institutions are responsible for
   ensuring fiscal transparency in India?",
  "retrieved_chunks": [
    "Enhancing Fiscal Transparency and Reporting in India",
    "India\s Fiscal Architecture",
    "Digitization and Services-Led Development",
    "South Asian Economic Focus",
    "Global Economic Prospects January 2021: Regional Overview
   SAR"
  ],
  "model_answer": "In India, the government institutions
   responsible for ensuring fiscal transparency include the
   Ministry of Finance, the Comptroller and Auditor General of
   India (CAG), and the Central Board of Direct Taxes (CBDT).",
  "relevance_scores": [0.7175, 0.6774, 0.3902, 0.3618, 0.3169],
  "hallucination_score": 0.6732,
  "pass_fail": "Pass"
}
```

Listing 6.3: Output JSON of RAG 3 from RAG pipeline

Chapter 7

CONCLUSION

This dissertation explored the integration of Large Language Models (LLMs) with Retrieval-Augmented Generation (RAG) pipelines to automate and enhance the analysis of India's national budget documents. By combining dense semantic retrieval with generative modeling, the system effectively interpreted policy-level queries and produced context-aware responses grounded in real government documents.

The experiments conducted demonstrated the system's capability to answer questions related to fiscal architecture, institutional transparency, and economic productivity with high semantic alignment and minimal hallucination. Evaluation metrics such as cosine similarity and hallucination scores validated the system's consistency and factuality.

Overall, the pipeline represents a practical advancement in AI-assisted public policy interpretation and fiscal transparency, serving as a foundation for future research, real-world deployment, and decision-support systems in governance.

Improvements

While the implemented system achieved its core objectives, several enhancements could further improve scalability, efficiency, and reliability:

1. **Limit Context to Top-N Most Relevant Sentences:**
   Rather than passing full retrieved text chunks to the LLM, the system can be optimized by extracting only the top-N most relevant sentences per chunk. This would:
   - Significantly reduce token count and inference cost.
   - Improve contextual focus and reduce noise.

Sentence-level relevance can be computed using cosine similarity between query and individual sentence embeddings.

2. **Use Fine-Tuned Small Language Models as Post-Hoc Validators:**
   Introduce a compact, fine-tuned verifier model (e.g., DistilBERT, Phi-2) trained on fiscal QA data. This model would:
   - Validate the LLM's output post-generation.
   - Flag or reject hallucinated or poorly aligned responses.

   This enables lightweight, low-latency hallucination filtering without relying solely on the primary LLM's internal confidence mechanisms.
3. **Cache and Index Embeddings Using FAISS or Annoy:**
   To improve retrieval scalability for large corpora, the system can use vector indexing tools like FAISS or Annoy to:
   - Accelerate top-k document/sentence retrieval in sub-linear time.
   - Handle thousands of embeddings efficiently without brute-force comparisons.

   This allows the system to scale to broader document bases without compromising speed.
4. **Corpus Expansion:**
   Incorporating state-level budgets, audit reports, and live financial indicators can make the system more robust and comprehensive.
5. **Interactive Front-End:**
   An intuitive query dashboard with explainability features (such as chunk highlights, score visualizations, and warnings) can improve usability for non-technical policy stakeholders.
6. **Multilingual Support:**
   Enabling queries in major Indian languages would democratize access to fiscal insights across linguistic groups.

In summary, this work demonstrates that a well-designed RAG system, augmented by structured improvements, can contribute significantly to fiscal transparency and intelligent policy discourse in India.

REFERENCES

[1] Yusen Yu. Finmem: A performance-enhanced llm trading agent with layered memory and character design. *arXiv preprint arXiv:2311.13743*, 2024.

[2] Hongxin Yang, Zihan Hu, and Zhiyuan Liu. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031*, 2023.

[3] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*, 2019.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019. Technical report.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[7] OpenAI. Gpt-4 technical report, 2023. `https://openai.com/research/gpt-4`.

[8] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Dawn Zou, Mantas Mazeika, Dawn Song, Jacob Steinhardt, et al. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2023.

[9] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. *arXiv preprint arXiv:2102.07350*, 2021.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[11] Zhijing Ji, Nayeon Lee, Jason Fries, Tao Yu, Dan Su, Yejin Zhang, et al. Survey of hallucination in natural language generation. *arXiv preprint arXiv:2302.03629*, 2023.

[12] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[14] pipiku915. Finmem-llm-stocktrading. `https://github.com/pipiku915/FinMem-LLM-StockTrading`, 2023.